

NASA-CR-204352

09/31/2

A HIGHLY ACCURATE VOIGT FUNCTION ALGORITHM

Z. SHIPPONY† and W. G. READ

Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109, U.S.A.

(Received 23 February 1993)

Abstract—A complex Voigt lineshape algorithm is presented whose maximum relative error over the complex plane is less than 1×10^{-8} . The algorithm consists of series, rational approximations and Gauss-Hermite integrations which makes it suitable as a general purpose software module for a wide variety of uses, including a Voigt function standard.

INTRODUCTION

The Voigt lineshape—a convolution between a Doppler velocity distribution and a first order collisional relaxation—is important for characterizing the frequency response in molecular spectroscopy and atmospheric radiative transfer. An algorithm for the complex Voigt function is needed for the polarized radiative transfer problem, where both the absorption and dispersion shapes must be included in the calculations. Furthermore, the real Voigt function frequency derivative is needed for applications requiring “linear” inversion of the lineshape function in terms of the linewidth parameters. The complex Voigt function is

$$W(z) = \frac{i}{\pi} \int_{-\infty}^{\infty} \frac{\exp\{-t^2\}}{z-t} dt, \quad (1)$$

where the customary definitions, $z = x - iy$ and $x = \sqrt{\ln 2}(v - v_0)/\gamma_D$ and $y = \sqrt{\ln 2}\gamma_L/\gamma_D$ are used. The collision and Doppler broadening widths are γ_L and γ_D , respectively. Equation 1 is related to the complex complementary error function ($\text{erfc}\{-iz\}$) according to

$$W(z) = \exp\{-z^2\} \text{erfc}\{-iz\} \quad (2)$$

for $y > 0$. Equation 2 cannot be evaluated in closed form. Consequently a number of algorithms have been developed for evaluating Eq. 2 which fall into three basic classes. A summary of these is given in a recent article by Schreier.¹

Within the three classes, those incorporating a combination of series, asymptotic and continued fraction expansions along with rational approximations and Gauss-Hermite integrations appear to provide the greatest accuracy with the minimum computation time. The algorithm described here incorporates these principles, and is more accurate and quicker than that given by Armstrong,² which was used as the accuracy standard by Schreier.¹ However, it is slower than some of the less accurate algorithms as that of Hui et al³ incorporating similar methods. As such, this algorithm is most useful when high accuracy is desired and as a Voigt function standard.

COMPUTATIONAL PROCEDURE

The approach employed here is an extension of Drayson's algorithm⁴ with improved series coefficients and rectangular regions. The computational regions are depicted in Fig. 1. The calculations within each region are computed as follows.

Region 1

The method used is the same as Drayson. $W(z)$ is evaluated with

$$W(z) = \exp\{-z^2\} + \frac{2i}{\sqrt{\pi}} F(z), \quad (3)$$

†To whom all correspondence should be addressed.

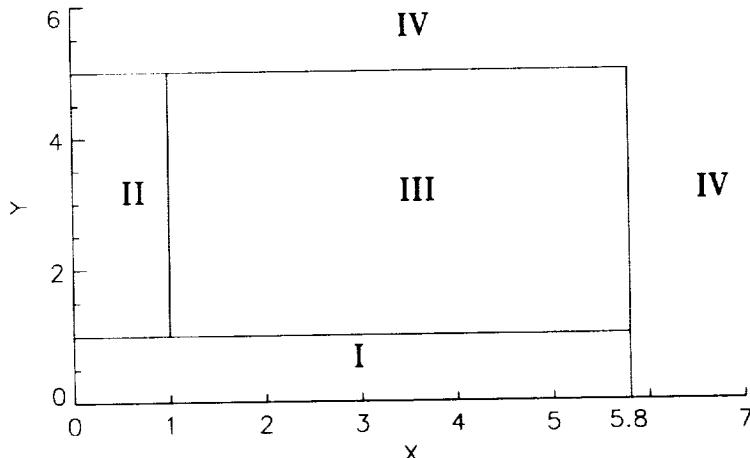


Fig. 1. Computational regions used for evaluating the Voigt function.

where

$$F(z) = \exp\{-z^2\} \int_0^z \exp\{t^2\} dt$$

is the Dawson function when $z (=x)$ is real. Expanding $F(z)$ in a Taylor series about x produces a recursion relation among the series coefficients,

$$d_0 = F(x), d_1 = 1 - 2xd_0, d_{n+1} = -\frac{2}{n+1} (xd_n + d_{n-1}) \quad \text{for } n = 1, 2, \dots \quad (4)$$

The Dawson function is evaluated for $x = 0.0$ to $x = 5.0$ in 0.5 steps using an adaptive Gauss-Legendre quadrature (with 10^{-16} accuracy). This forms the basis for a Taylor series approximation to Dawson's function for any x . The Dawson function values are presented in Table 1. This differs from Ref. 4, where a Chebyshev expansion is used as an approximation to Dawson's function.

After $F(x)$ is evaluated, Drayson's Taylor series (in y) is used to obtain the final complex Voigt function. The series order is determined by an empirical function which differs from Drayson's as a result of the different region being used here. The function used here is best described by the FORTRAN code fragment,

```

if (x .lt. 3.0) then
    j = 9
    q = 16.0
else if (x .lt. 4.0) then
    j = 8
    q = 17.0
else
    j = 7
    q = 14.0
endif
n = min(j + nint(q * (y - 0.1)/0.9), 25)

```

Table 1. Dawson function values between $x = 0.0$ and $x = 5.0$.

x	$F(x)$	x	$F(x)$
0.0	0.0000000000000000	3.0	0.1782710306105583
0.5	0.4244363835020223	3.5	0.1496215930807565
1.0	0.5380795069127685	4.0	0.1293480012360052
1.5	0.4282490710853986	4.5	0.1140886102268250
2.0	0.3013403889237919	5.0	0.1021340744242768
2.5	0.2230837221674355		

where n is the number of terms and $\text{nint}(r)$ is a function that computes $r + 0.5$ prior to integer conversion.

Region 2

The six coefficient rational approximation given by Hui et al³ is used.

Region 3

A complex Taylor series expansion of $W(z)$ whose coefficients are based upon exact (10^{-16} relative error) function evaluations of the Voigt in region 3. The gridding covers $1.0 \leq x \leq 5.0$ and $1.0 \leq y \leq 5.0$ in 1.0 steps in each axis. Complex Voigt [$W(z)$] values are given in Table 2. These are converted to Taylor coefficients, which are used as an approximation to the complex Voigt function, by making use of the recursive expression of $W(z)$ derivatives as described in Ref. 5.

The complex Voigt at (x, y) is evaluated by locating the nearest (x_0, y_0) grid point in the table and using the Taylor series expansion appropriate for that point. Fewer than 14 Taylor series terms guarantee 10^{-8} accuracy.

Region 4

In this region an eight point Gauss-Hermite quadrature is used.

ACCURACY AND COMPUTING TIMES

The speed and accuracy of this approach is contrasted with some widely used algorithms. A reference Voigt function was generated from an adaptive Gauss-Legendre quadrature whose accuracy was maintained at 10^{-16} . Figures 2-5 show the accuracy of this algorithm and those of

Table 2. $W(z)$ function values between $1.0 \leq x \leq 5.0$ and $1.0 \leq y \leq 5.0$.

Real($W(z)$)			
y - axis values			
x	1.0	2.0	3.0
1.0	0.3047442052569126	0.2184926152748907	0.1642611363929863
2.0	0.1402395813662779	0.1479527595120158	0.1307574696698486
3.0	0.0653177772890470	0.0927107664264433	0.0964025055830445
4.0	0.0362814564899886	0.0596869296104459	0.0697909616496483
5.0	0.0230031325940600	0.0406436763334944	0.0512259965673866
Real($W(z)$)			
y - axis values			
x	4.0	5.0	
1.0	0.1298881599308405	0.1067977383980653	
2.0	0.1121394779021160	0.0964981126066414	
3.0	0.0909339041947653	0.0829877379769018	
4.0	0.0715704334263653	0.0692362095804914	
5.0	0.0559973771425239	0.0569654398881770	
Imaginary($W(z)$)			
y - axis values			
x	1.0	2.0	3.0
1.0	0.2082189382028316	0.0929978093926019	0.0501971351352486
2.0	0.2222134401798991	0.1311797170842179	0.0811126504774567
3.0	0.1739183154163490	0.1283169622282616	0.0912363260042188
4.0	0.1358389510006551	0.1132100561244882	0.0893400002403649
5.0	0.1103328325535800	0.0979873111565719	0.0828369131719072
Imaginary($W(z)$)			
y - axis values			
x	4.0	5.0	
1.0	0.0307788608170588	0.0206040887146843	
2.0	0.0534889938529669	0.0373516531563688	
3.0	0.0655923305279143	0.0483893652029131	
4.0	0.0693745186137715	0.0540702270359291	
5.0	0.0682948856449228	0.0558387427753910	

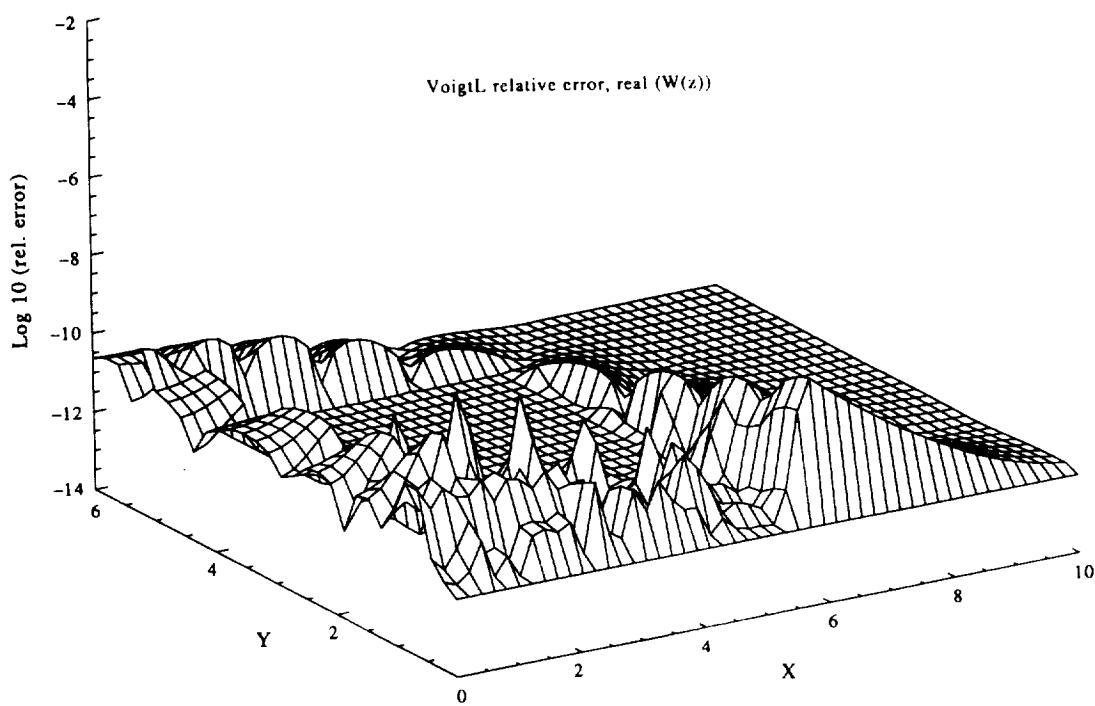


Fig. 2. Logarithm of relative error: this paper's algorithm vs a Voigt standard.

Armstrong,² Drayson,⁴ and Hui³ for the real part of the Voigt function. The two dimensional grid considered 50 x values spanning $0.0 \leq x \leq 10.0$, and 30 y values covering $0.0 \leq y \leq 6.0$ for a total of 1500 points. The Hui algorithm employed Karp's modification given by Schreier.¹ All calculations used double precision arithmetic. The peak relative error obtained from our algorithm is less than 1 part in 10^8 greatly exceeding the accuracy of all others. The average and maximum

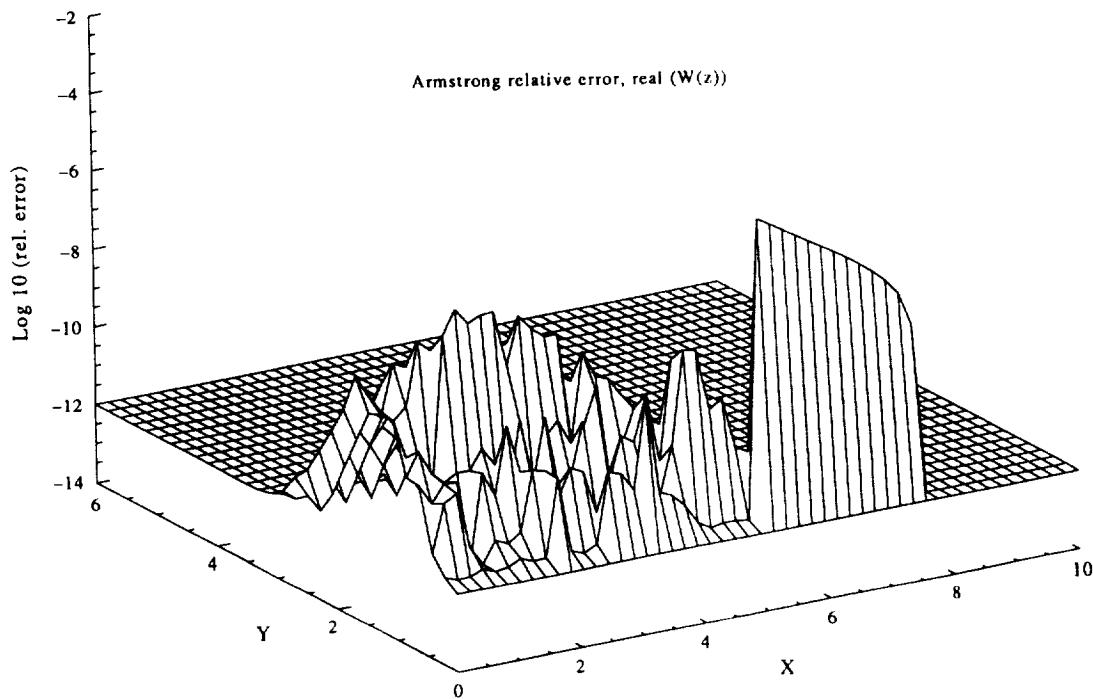


Fig. 3. Logarithm of relative error: the Armstrong algorithm vs a Voigt standard.

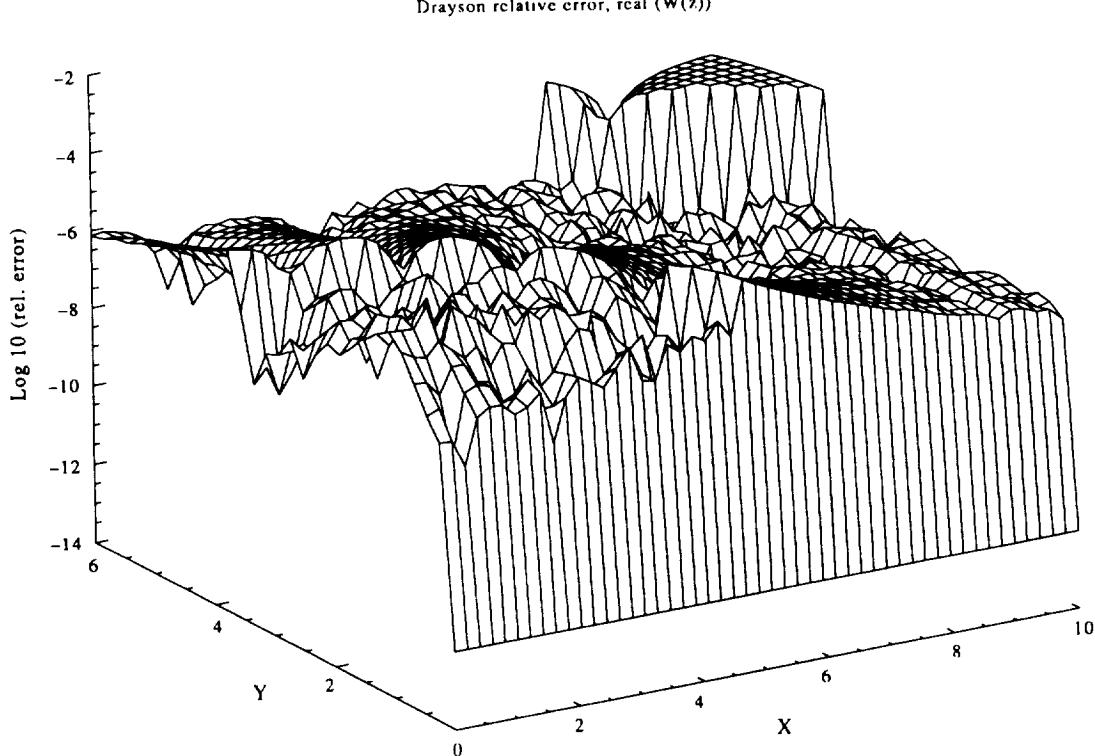


Fig. 4. Logarithm of relative error: the Drayson algorithm vs a Voigt standard.

relative error and average computer time per point are presented in Table 3. The programs, coded in VAX FORTRAN as subroutines with no vectorization, were run on a MIPS RISC 3000 computer. Our algorithm is 1.9 and 2.8 times slower than Hui's and Drayson's algorithms respectively, and 1.3 times faster than Armstrong's. FORTRAN code for the algorithm is in the Appendix.

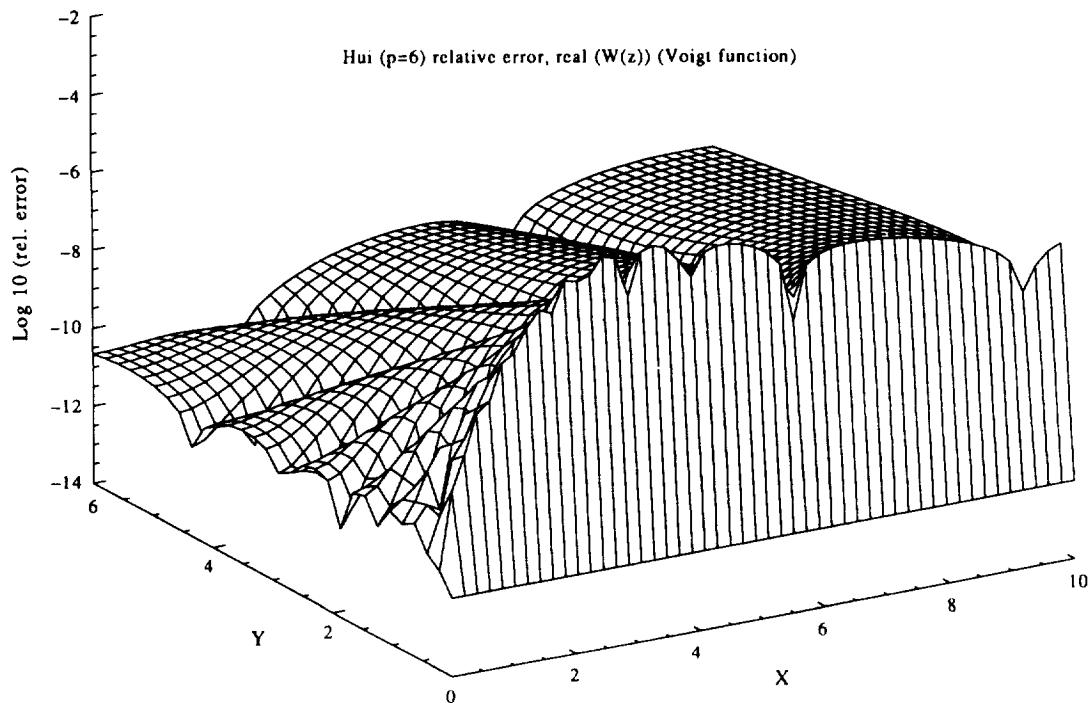


Fig. 5. Logarithm of relative error: the Hui algorithm vs a Voigt standard.

Table 3. Comparison of accuracy and times for different algorithms.

Algorithm	Peak Relative Error	Average Relative Error	Average time per Point (μ sec)
This Work	5×10^{-9}	6×10^{-11}	53.3
Armstrong ²	1×10^{-4}	2×10^{-9}	68.6
Drayson ⁴	6×10^{-4}	1×10^{-6}	19.0
Hui ³	6×10^{-5}	4×10^{-7}	28.0

CONCLUSION

A Voigt algorithm is presented that is simple to implement, yet highly accurate for all combinations of x and y . The complex Voigt function is computed, so the algorithm can be used for derivatives and molecular line dispersion in addition to absorption computations. For applications where speed is of primary concern, Drayson's or Hui's algorithm (which compromise accuracy for speed) are the methods of choice especially if other uncertainties exceed one part in 10^{-4} . When high accuracy is needed, this algorithm's greater accuracy is desirable (typically 4–5 orders of magnitude better than the faster routines). This accuracy is achieved with a faster algorithm than that of Armstrong² which has been recognized as the most accurate among the single value Voigt function routines.¹

Acknowledgements —The authors thank Drs Joe W. Waters and William Lahoz for helpful comments. This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

REFERENCES

1. F. Schreier, *JQSRT* **48**, 743 (1991).
2. B. H. Armstrong, *JQSRT* **7**, 61 (1967).
3. A. K. Hui, B. H. Armstrong, and A. A. Wray, *JQSRT* **19**, 509 (1978).
4. S. R. Drayson, *JQSRT* **16**, 611 (1976).
5. M. Abramowitz and A. Stegun, *Dover*, p. 298, New York, NY (1970).

APPENDIX

Voigt Subroutine FORTRAN Source Code

```

Subroutine VoigtL(x,y,u,v)
c*****                                         *****
c                                         *
c Computes the Voigt function: Integral from - to + infinity of:   *
c                                         *
c      (y/Pi)*Exp(-t*t)/(y*y+(x-t)*(x-t)) dt                         *
c                                         *
c and:                                         *
c      (1/Pi)*(x-t)*Exp(-t*t)/(y*y+(x-t)*(x-t)) dt                         *
c                                         *
c Developed by Z. Shippony, Jet Propulsion Laboratory (November/1990)   *
c                                         *
c Here:                                         *
c      x = sqrt(ln 2) * (v - v0) / aD      (x >= 0.0)                   *
c      y = sqrt(ln 2) * aL / aD            (y >= 0.0)                   *
c                                         *
c Where v is the wave number, v0 is the line center wave number, aL   *
c is the Lorentzian line half-width and aD is the Doppler line           *
c half-width.                                         *
c*****                                         *****
Implicit real*8 (a-h,o-z)
Parameter (xf=5.8D0)
Parameter (ys=1.0D0, yf=5.0D0)
if(y.gt.yf.or.x.gt.xf) then
  Call VoigtH(x,y,u,v)          ! Region 4
else
  if(y.lt.ys) then
    if(x.ge.ys) then
      Call VoigtR3(x,y,u,v)      ! Region 3
    else
      Call VoigtR2(x,y,u,v)      ! Region 2
    end if
  end if
end if

```

```

        else
            Call VoigtR2(x,y,u,v)      ! Region 2
        endif
    else
        Call VoigtR1(x,y,u,v)      ! Region 1
    endif
endif
Return
End

c-----
Subroutine VoigtR1(x,y,u,v)
c*****
c Computes the Voigt function for the region (xs <= x <= xf, y <= ys) *
c (Here xs = 0.0, xf = 5.8, ys = 1.0) *
c *
c Based on the algoritm described in: *
c
c     "Rapid Computation of the Voigt Profile" by S. R. Drayson, *
c     JQSRT, Vol. 16, 611, (1976). *
c
c (Modified by Z. Shippony, Jet Propulsion Laboratory, November/1990) *
c
c*****
Implicit none
Real*8 xtf,delx,Tiny
Integer maxd,maxr,maxf
Parameter (xtf = 5.0d0, maxd = 11, maxr = 25, maxf = 13)
Parameter (delx = xtf / (maxd-1), Tiny = 1.0d-12)
Real*8 Dowson(maxd),Fcoef(maxf,maxd),hn(maxd), q, p, y2, EpsLn,
&      dyr, dyi, zi, zr, ri(maxr), v, u, dx, x, y, twovspi
Logical First
Integer*4 i, j, n
SAVE First, ri, hn, Fcoef, EpsLn
Data First/.True./
Data twovspi /1.1283791670955125739d0/      ! 2.0 / Sqrt(Pi)
c Dawson function (F(z)) table, for y = 0, x = 0.0 to: 5.0d0  dx: 0.5
Data Dowson / 0.0d0 , 4.244363835020223d-1, 5.380795069127685d-1,
& 4.282490710853986d-1, 3.013403889237919d-1, 2.230837221674355d-1,
& 1.782710306105583d-1, 1.496215930807565d-1, 1.293480012360052d-1,
& 1.140886102268250d-1, 1.021340744242768d-1 /
if (First) then
    First = .False.
    EpsLn = Dlog(1.0d-8)
    do 10 i = 1, maxr
        ri(i) = -2.0d0 / Dfloat(i)
10 Continue
c Compute Dawson's function at mesh points
do 30 i = 1, maxd
    u = Dfloat(i - 1) * delx
    hn(i) = u
    p = Dowson(i)
    q = 1.0d0 - 2.0d0 * u * p
    Fcoef(1,i) = q
    do 20 j = 2, maxf
        v = (u * q + p) * ri(j)
        Fcoef(j,i) = v
        p = q
        q = v
20 Continue
30 Continue
endif
c Compute Dawson's function at x from Taylor series
y2 = y * y
j = Int(x / delx + 0.5) + 1
n = min0(j, maxd)
dx = x - hn(n)
p = Dabs(dx)
if(p.le.Tiny) then
    u = Dowson(n)                      ! F(x)
else
    u = maxf
    v = 0.0d0

```

```

if(p.lt.1.0) u = EpsLn / Dlog(p)
j = int(u + 0.5)
i = min0(j, maxf)
do 40 j = i, 1, -1
  v = dx * v + Fcoef(j,n)

40 Continue
  u = dx * v + Dowson(n)           ! F(x)
endif
if(y.le.Tiny) then
  v = twovspi * u
  u = Dexp(-x*x)
  Return
endif

c** Taylor series expansion about y = 0.0
c Compute no. of terms to use before truncation:
  if(x.lt.3.0) then
    j = 9
    q = 16.0
  else if(x.lt.4.0) then
    j = 8
    q = 17.0
  else
    j = 7
    q = 14.0
  endif
  p = q * (y - 0.1) / 0.9
  i = j + Int(p + 0.5)
  n = min0(i, maxr)               ! Number of terms in the series
  q = u                           ! F(x)
  v = 1.0d0 - 2.0d0 * x * u     ! F'(x)
  zr = u
  zi = y * v
  dyr = 0.0d0
  dyi = y                         ! dy = Cmplx(0.0,y)
  do 50 i = 2, n
    p = dyr
    dyr = -dyi * y
    dyi = p * y                   ! = dy^n
    u = (x * v + q) * ri(i)
    zr = zr + u * dyr
    zi = zi + u * dyi
    q = v
    v = u

50 Continue
  p = -2.0d0 * x * y
  q = Dexp(y2 - x * x)
  u = q * Dcos(p) - twovspi * zi ! Exp(-z*z) + 2*i*F(z)/Sqrt(Pi)
  v = q * Dsin(p) + twovspi * zr
  Return
End

c-----Subroutine VoigtR2(x,y,u,v)
c*****Computes the Voigt function for the region:
c      (0.0 <= x < 1.0, 1.0 <= y <= 5.0)
c
c Hui's method was coded according to:
c
c "Rapid computation of the Voigt and complex error function"
c
c by: A.K. Hui, B.H. Aramstrong and A.A. Wray,
c JQSRT, Vol 19, 506, (1978).
c
c Here:
c           x = sqrt(ln 2) * (v - v0) / aD   (x >= 0.0)
c           y = sqrt(ln 2) * aL / aD          (y >= 0.0)
c
c Where v is the wave number, v0 is the line center wave number, aL
c is the Lorentzian line half-width and aD is the Doppler line
c half-width.
c
```

```

Implicit real*8 (a-h,o-z)
Dimension a(7),b(7)
Complex*16 z,az,dz,w
c Hui's (p = 6) rational approximation coefficients:
  Data a / 122.607931777104326d0, 214.382388694706425d0,
  &          181.928533092181549d0, 93.155580458138441d0,
  &          30.180142196210589d0, 5.912626209773153d0,
  &          0.564189583562615d0 /
  Data b / 122.607931773875350d0, 352.730625110963558d0,
  &          457.334478783897737d0, 348.703917719495792d0,
  &          170.354001821091472d0, 53.992906912940207d0,
  &          10.479857114260399d0 /
c Region: (0.0 <= x < 1.0, 1.0 <= y <= 5.0)
c Hui (p = 6) algorithm:
  z = Dcmplx(y,-x)
  az = a(1)+z*(a(2)+z*(a(3)+z*(a(4)+z*(a(5)+z*(a(6)+z*a(7)))))))
  dz = b(1)+z*(b(2)+z*(b(3)+z*(b(4)+z*(b(5)+z*(b(6)+z*(b(7)+z)))))))
  w = az / dz
  u = Dreal(w)
  v = Dimag(w)
  Return
End
c-----
c----- Subroutine VoigtR3(x,y,u,v)
c***** Computed the Voigt function for the region:
c      (xs <= x <= xf, ys <= y <= yf)
c
c (Here xs = 1.0, xf = 5.8, ys = 1.0, yf <= 5.0)
c
c This Code was developed by Z. Shippony,
c Jet Propulsion Laboratory (November/1990)
c
c Here:
c      x = sqrt(ln 2) * (v - v0) / aD      (x >= 0.0)
c      y = sqrt(ln 2) * aL / aD            (y >= 0.0)
c
c Where v is the wave number, v0 is the line center wave number, aL
c is the Lorentzian line half-width and aD is the Doppler line
c half-width.
c
c*****
Implicit real*8 (a-h,o-z)
Parameter (xs=1.0D0, xf=5.8D0)
Parameter (ys=1.0D0, yf=5.0D0)
Parameter (dx=1.0D0, dy=1.0D0)
Parameter (maxx = 5, maxy = 5, nmax = 13)
Parameter (Tiny = 1.0d-12, Eps = 1.0d-8)
Logical First
Dimension wr(maxx,maxy),wi(maxx,maxy),Fn(nmax)
Complex*16 wd(nmax,maxx,maxy),z,w,az,dz,ds,Sum,srm1
SAVE First, dxh, dyh, wd
c Table for w(z) = w(x,y)
c x = 1.0,..., 5.0, Step: 1.0
c y = 1.0,..., 5.0, Step: 1.0
  Data ((wr(i,j),j=1,5),i=1,5)/
  & 3.047442052569126D-1, 2.184926152748907D-1, 1.642611363929863D-1,
  & 1.298881599308405D-1, 1.067977383980653D-1, 1.402395813662779D-1,
  & 1.479527595120158D-1, 1.307574696698486D-1, 1.121394779021160D-1,
  & 9.649811260664139D-2, 6.53177728904696D-2, 9.271076642644334D-2,
  & 9.640250558304454D-2, 9.093390419476534D-2, 8.298773797690175D-2,
  & 3.628145648998864D-2, 5.968692961044590D-2, 6.979096164964831D-2,
  & 7.157043342636532D-2, 6.923620958049143D-2, 2.300313259405996D-2,
  & 4.064367633349437D-2, 5.122599656738663D-2, 5.599737714252388D-2,
  & 5.69654398817697D-2 /
  Data ((wi(i,j),j=1,5),i=1,5)/
  & 2.082189382028316D-1, 9.299780939260188D-2, 5.019713513524858D-2,
  & 3.077886081705883D-2, 2.060408871468425D-2, 2.222134401798991D-1,
  & 1.311797170842179D-1, 8.111265047745665D-2, 5.348899385296694D-2,
  & 3.735165315636876D-2, 1.739183154163490D-1, 1.283169622282616D-1,
  & 9.123632600421876D-2, 6.559233052791429D-2, 4.838936520291309D-2,
  & 1.358389510006551D-1, 1.132100561244882D-1, 8.934000024036491D-2,
  & 6.937451861377146D-2, 5.407022703592907D-2, 1.103328325535800D-1,
```

```

& 9.798731115657190D-2, 8.283691317190719D-2, 6.829488564492278D-2,
& 5.583874277539103D-2 /
Data First/.True./
Data srml/(0.0d0,1.0d0)/
Data SPi/1.7724538509055160273d0/ ! Sqrt(Pi)
if(First) then
c Compute all the necessary derivatives of W(z) at mesh points:
c (For the recursive expression of W(z) derivatives, see "Handbook of
c Mathematical functions", M. Abramowitz & A. Stegun, Dover publications,
c Nov. 1970, pp. 298, Eq.: 7.1.21)
First = .False.
dxh = 0.5d0 * dx
dyh = 0.5d0 * dy
do 10 i = 1, nmax
    Fn(i) = -2.0d0 / Dfloat(i+1)
10   continue
xx = xs - dx
do 40 ix = 1, maxx
    xx = xx + dx
yy = ys - dy
do 30 iy = 1, maxy
    yy = yy + dy
z = Dcmplx(xx,yy)
w = Dcmplx(wr(ix,iy),wi(ix,iy))
ds = 2.0d0 * (srml / SPi - z * w)
wd(1,ix,iy) = ds
do 20 i = 1, nmax-1
    az = Fn(i) * (w + z * ds)
    wd(i+1,ix,iy) = az
    w = ds
    ds = az
20   continue
30   continue
40   continue
endif
c Region: (xs <= x <= xf, ys <= y <= yf)
c Using Taylor series expansion for W(z).
ix = 1 + Int((x-xs) / dx)
xx = xs + (ix - 1) * dx
if(x-xx.gt.dvh) ix = ix + 1
if(ix.gt.maxx) ix = maxx
xx = xs + (ix - 1) * dx
iy = 1 + Int((y-ys) / dy)
yy = ys + (iy - 1) * dy
if(y-yy.gt.dyh) iy = iy + 1
if(iy.gt.maxy) iy = maxy
yy = ys + (iy - 1) * dy
u = wr(ix,iy)
v = wi(ix,iy)
p = x - xx
q = y - yy
if(Dabs(p).lt.Tiny.and.Dabs(q).lt.Tiny) Return
dz = Dcmplx(p,q)
q = Dsqrt(p * p + q * q)
p = Eps / Cdabs(wd(1,ix,iy))
p = Dlog(p) / Dlog(q)
n = 1 + min0(nmax-1,Int(p))
Sum = Dcmplx(0.0d0,0.0d0)
do 50 i = n, 1, -1
    Sum = dz * Sum + wd(i,ix,iy)
50   continue
w = dz * Sum
u = u + DReal(w)
v = v + DIImag(w)
Return
End
C-----
Subroutine VoigtH(x,y,u,v)
*****
c Computes the complex Voigt function: (i/pi)*integral from - to + infinity *
c of : exp(-t*t)/(z - t) dt, where z=(x+iy), x, y >= 0. Using Gauss-Hermite *

```

```

c Quadrature (8 points). The real part is u, the integral of:
c
c           (y/pi)*Exp(-t*t)/(y*y + (x-t)*(x-t)) dt
c
c The imaginary part is v, the integral of:
c
c           (1/pi)*(x-t)*Exp(-t*t)/(y*y + (x-t)*(x-t)) dt
c
c ****
Implicit Real*8 (a-h,o-z)
Parameter (N = 4)
Dimension Gx(N), Gw(N)
c Note: The roots are both positive and negative !
Data Gx/
& 3.8118699020732211685d-1, 1.1571937124467801947d0,
& 1.9816567566958429259d0 , 2.9306374202572440192d0 /
Data Gw/
& 6.6114701255824129103d-1, 2.0780232581489187954d-1,
& 1.7077983007413475456d-2, 1.9960407221136761921d-4 /
Data Pi / 3.1415926535897932385d0 / ! Pi
y2 = y * y
Sumu = 0.0d0
Sumv = 0.0d0
do 10 i = 1, N
  t = Gx(i)
  xpt = x + t
  xmt = x - t
  fm = 1.0d0 / (y2 + xmt * xmt)
  fp = 1.0d0 / (y2 + xpt * xpt)
  Sumu = Sumu + Gw(i) * (fm + fp)
  Sumv = Sumv + Gw(i) * (xmt * fm + xpt * fp)
10 Continue
u = y * Sumu / Pi
v = Sumv / Pi
Return
End

```

